List prediction for motion planning: Case studies

Abhijeet Tallavajhula, Sanjiban Choudhury

CMU-RI-TR-15-25

October 2015

Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

There is growing interest in applying machine learning to motion planning. Potential applications are predicting an initial seed for trajectory optimization, predicting an effective heuristic for search based planning, and even predicting a planning algorithm for adaptive motion planning systems. In these situations, providing only a *single* prediction is unsatisfactory. It leads to many scenarios where the prediction suffers a high loss. In this paper, we advocate *list* prediction. Each predictor in a list focusses on different regions in the space of environments. This overcomes the shortcoming of a single predictor, and improves overall performance. A framework for list prediction, ConseqOpt, already exists. Our contribution is an extensive domain-specific treatment. We provide a rigorous and clear exposition of the procedure for training a list of predictors. We provide experimental results on a spectrum of motion planning applications. Each application contributes to understanding the behavior of list prediction. We observe that the benefit of list prediction over a single prediction is significant, irrespective of the application.

Contents

1 Introduction

There have been a number of efforts to introduce procedures from machine learning to motion planning. These procedures offer the ability to adapt motion planning algorithms to a specific environment. Adaptation is performed in a data-driven manner, and past experience in the form of solved motion planning environments can be leveraged to train the learning procedure.

In this work, we consider adaptation in the form of selecting from a fixed set of options. Assume we have a library of elements, and each element is an option for a motion planning algorithm. Given an environment, we want to predict which element to use. For instance, in a trajectory optimization algorithm, the library elements are seed trajectories, and the environment is the configuration of obstacles. The objective is to find a learning procedure that has good prediction performance, i.e, how well can the learnt procedure predict the best element in the library?

Machine learning procedures provide guarantees on expected performance, which is the average performance over a probability distribution of environments. A predictor trained on this distribution has interesting behavior. It predicts the correct element on most environments, but has extremely low performance on environments which are infrequent. This is particularly unsatisfactory for motion planning, which demands good prediction performance on all environments.

In this work, we do not tease a worst-case performance guarantee out of learning theory. Instead, we advocate a procedure that takes steps towards meeting the motion planning performance demand—list prediction. This involves training not one, but multiple predictors. They are trained sequentially. Importantly, they focus on different environments. As illustrated in Fig. 1, the second predictor will focus on environments where the first predictor had low performance. The third will focus on environments where the first two had low performance, and so on.

From a learning viewpoint, a list of predictors will always perform better than a single predictor. A list also addresses the motion planning concern of not only how, but also where performance is low—the second predictor will attempt to do well on the infrequent environments which the first predictor ignored. A list is also compatible with motion planning applications which have the scope to evaluate multiple options in parallel, and select the best one. The size of the list, henceforth called the budget, can be chosen to reflect the computational resources available to the motion planning systems to make a decision within a time period.

List prediction is not a new topic, having been introduced to motion planning as ConseqOpt [6]. We use ConseqOpt as a base, but make the following contributions

1. Clear derivation of the training procedure for list prediction in the framework of loss-sensitive multiclass classification.

2. Analyzing the behavior of predictors at different levels across a spec-

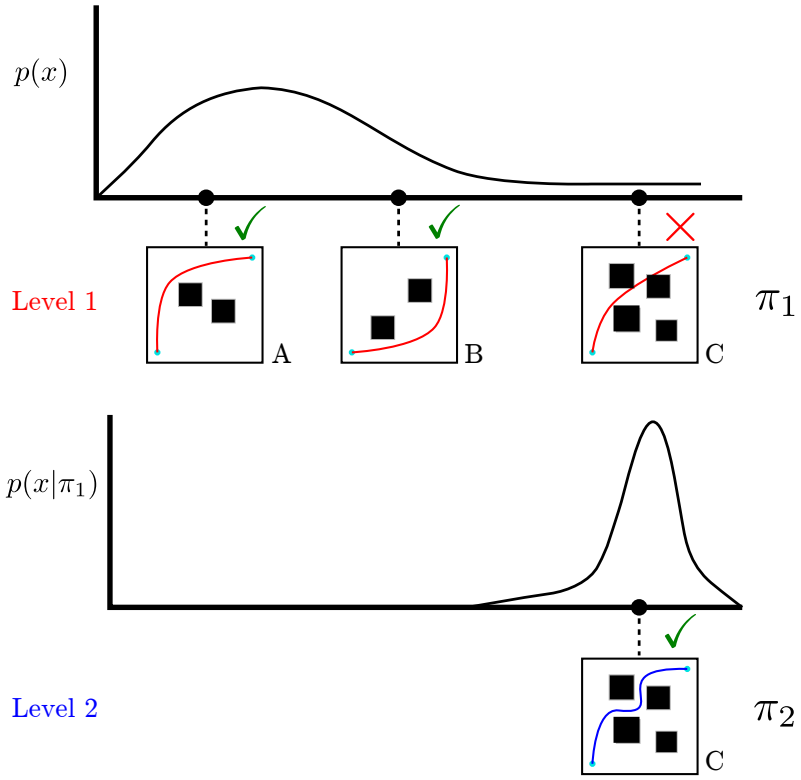

Figure 1: An illustration of how list prediction works for seed prediction. The first level predictor, π_1, performs well on environments such as A and B which are likely under the distribution $p(x)$. It performs poorly on infrequent environments like C . At the second level, unsolved environments are likely under the distribution $p(x|\pi_1)$. C is solved by predictor π_2.

trum of motion planning applications.

The overview of the paper is as follows—in Section 2 we describe the problem formally, at the end of which it is evident that training a list of predictors is justified. Section 3 is devoted to implementation, where we spend time discussing exactly how prediction is performed. Some properties of list prediction are highlighted in Section 4. In Sections 5 - 8, we explore the operation of ConseqOpt, developing intuition and visualizing predictions. Our experiments are on the entire spectrum of planning problems—from predicting seeds for trajectory optimization, to heuristics for search based planners, to planning algorithms for adaptive motion planning systems. We believe that our insights will benefit any effort undertaken in applying prediction to motion planning, clarifying understanding and arguing for the well-grounded procedure of list prediction.

2 Formulation

2.1 Element prediction

Let $x \in X$ denote a motion planning environment sampled from a distribution $p(x)$. We assume a library $\mathcal{L}$ of L elements is given, such that $\mathcal{L} = \{\xi_j\}, j = 1\colon L,\ \xi_j \in \Xi$. The cost of element ξ in environment x is a scalar, denoted by $c(x, \xi)$. If planning time was not a constraint, we would step through the library and pick the lowest cost element, $\arg\min_{\xi \in \mathcal{L}} c(x, \xi)$. Instead, we want to predict an element which will have low cost. Since prediction can only do as well as the best element in the library, we define loss as a relative cost. The loss of predicting element ξ_j is

$$l(x, \xi_j) = c(x, \xi_j) - \min_{\xi \in \mathcal{L}} c(x, \xi) \tag{1}$$

A predictor π, belonging to a class of predictors Π, takes an environment as input and outputs a library element, i.e, $\pi : X \to \Xi$. The loss of a predictor is

$$l(x, \pi) = c(x, \pi(x)) - \min_{\xi \in \mathcal{L}} c(x, \xi) \tag{2}$$

Finally, the risk of π is the expected loss

$$R(\pi) = \int l(x, \pi) p(x) dx \tag{3}$$

We would like to find the minimum risk predictor. Assume we have training data in the form of environments and element costs, $\mathcal{D} = \{(x_i, c(x_i, \xi_j))\}, i = 1\colon N, j = 1\colon L,\ x_i \sim p(x),\ \xi_j \in \mathcal{L}$. The objective can then be stated as finding the predictor which minimizes the empirical risk

$$\hat{R}(\pi) = \frac{1}{N} \sum_{i=1}^{N} l(x_i, \pi) \tag{4a}$$

$$\pi = \arg\min_{\tilde{\pi} \in \Pi} \hat{R}(\tilde{\pi}) \tag{4b}$$

Observe that predicting one of L elements given x is like classifying x into one of L classes. So π is a multiclass classifier. The 'correct' class is the minimum loss element, but misclassification losses differ. Specifically, finding π is a case of loss-sensitive multiclass classification[1].

The loss l is non-convex, which makes it difficult to directly minimize the empirical risk. The solution approach detailed in [2], which we follow, is to replace l with a convex function, known as the surrogate loss. This is a well-known procedure in learning. To recall the simple case of binary classification, the $0-1$ loss is also non-convex. Binary classification is solved

[1] Called *cost*-sensitive classification in the learning literature. Since we use costs to refer to element costs, we use the term *loss*-sensitive classification.

by replacing the loss with a convex surrogate. The step here can be thought of as a more general version of using surrogates. As per the analysis in [2], minimizing the convex surrogate risk implies minimizing the true risk. The predictor π is defined in terms of a scoring function. Let $s(x, \xi) \in \mathbb{R}$ be a function which assigns a score to library element ξ in environment x. The predictor then picks the element with the highest score

$$\pi(x) = \arg\max_j s(x, \xi_j) \tag{5}$$

Risk will be minimized in terms of the scoring function s, and therefore π. The predictor space Π also depends on the space of scoring functions. For any environment, the scores are required to be centered over the library elements. Intuitively, this means that the scores are required to be well-behaved. Otherwise we would be free to assign arbitrarily high scores to low-loss elements or arbitrarily low scores to high-loss elements

$$\sum_{j=1}^{L} s(x, \xi_j) = 0, \; \forall \, x \tag{6}$$

The last ingredient is $\psi(t) \in \mathbb{R}$, a convex function of t. The surrogate loss is then defined as

$$l_\psi(x, s) = \sum_{j=1}^{L} l(x, \xi_j) \psi(s(x, \xi_j)) \tag{7}$$

The scores $s(x, \xi)$ don't enter l_ψ directly, but through ψ. The surrogate loss is large when either the true loss $l(x, \xi_j)$ or $\psi(s(x, \xi_j))$ is large. Intuitively, it encourages high scores for the low-loss elements. The empirical surrogate risk is defined in terms of the surrogate loss. With the following shorthand: $l_{ij} = l(x_i, \xi_j)$, $s_{ij} = s(x_i, \xi_j)$, the transformed optimization is

$$\hat{R}_\psi(s) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{L} l_{ij} \psi(s_{ij}) \tag{8a}$$

$$s = \arg\min_{\tilde{s}} \; \hat{R}_\psi(\tilde{s}) \tag{8b}$$

We detail the choices of the convex surrogate $\psi(t)$, scorer $s(x, \xi)$, and the optimization scheme in Section 3.

2.2 List prediction

An algorithm for list prediction, ConseqOpt, was presented in [6]. We broadly follow their treatment. Instead of a single predictor, we now want to find a list of predictors. The length of the list, or budget B, is fixed and decided by computational resources. The list of predictors is denoted by $\langle \pi^1 \colon \pi^B \rangle$. The list of elements predicted for environment x is $\langle \pi^1(x) \colon \pi^B(x) \rangle$.

We use superscripts for levels in the list. Definitions in 2.1 are extended to lists. The loss in (2) is extended to take a list of predictors as argument

$$l(x, \langle \pi^1 \colon \pi^B \rangle) = \min_k c(x, \pi^k(x)) - \min_{\xi \in \mathcal{L}} c(x, \xi) \tag{9}$$

Observe that the loss only cares about the lowest cost element in the predicted list, $\min_k c(x, \pi^k(x))$. The risk of a list of predictors is the expected loss

$$R(\langle \pi^1 \colon \pi^B \rangle) = \int l(x, \langle \pi^1 \colon \pi^B \rangle) p(x) dx \tag{10}$$

The increased power of a list of predictors over a single predictor may seem to be offset by the difficult task of finding the optimal list. The optimal list requires a search over all lists of length B

$$\langle \pi^{1*} \colon \pi^{B*} \rangle = \arg \min_{\langle \tilde{\pi}^1 \colon \tilde{\pi}^B \rangle \in \Pi^B} R(\langle \tilde{\pi}^1 \colon \tilde{\pi}^B \rangle) \tag{11}$$

However, the risk is a monotone supermodular function. We don't spend time on supermodular and submodular functions here, but see [6] for details. The implication of this property is that there exists a simple algorithm for finding a near-optimal list: greedy selection. Selecting the first predictor π^1 is exactly the minimization in (4b). The second predictor is selected as $\pi^2 = \min_{\tilde{\pi} \in \Pi} R(\langle \pi^1, \tilde{\pi} \rangle)$, or minimizing the risk after fixing the predictor at the first level. In general, the greedy procedure is

$$R^k(\pi) = R(\langle \pi^1 : \pi^{k-1}, \pi \rangle) \tag{12a}$$

$$\pi^k = \arg \min_{\tilde{\pi} \in \Pi} R^k(\tilde{\pi}),\ k = 1 \colon B \tag{12b}$$

We refer to $R^k(\pi)$ as the risk at level k. It is a function of $\langle \pi^1 \colon \pi^{k-1} \rangle$. We also use the shorthand $l^k_{ij} = l(x_i, \langle \pi^1 \colon \pi^{k-1}, \xi_j \rangle)$ for the losses at the level k. They calculate the loss of element ξ_j, given the list of predictors $\langle \pi^1 \colon \pi^{k-1} \rangle$. For example, if one of the earlier predictors has already predicted the lowest loss element for environment x_i, then it does not matter which element is predicted at level k, and all the marginal losses will be zero, $l^k_{ij} = 0\ \forall j$. We again use the convex relaxation, optimizing the empirical surrogate risks at each level. The predictor π^k is defined in terms of a scoring function s^k at that level

$$\hat{R}^k_\psi(s) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{L} l^k_{ij} \psi(s_{ij}) \tag{13a}$$

$$s^k = \arg \min_{\tilde{s}} R^k_\psi(\tilde{s}) \tag{13b}$$

$$\pi^k(x) = \arg \max_j s^k(x, \xi_j),\ k = 1 \colon B \tag{13c}$$

We close this section with an observation about the risks. If X is the space of environments, let $X|\neg 1$ be the space of environments where loss is non-zero under π^1. Equivalently, $X|\neg 1$ is the space of unsolved environments at level 2. More generally, $X|\neg 1\colon \neg(k-1)$ is the space of environments where loss is nonzero under the list $\langle \pi^1 \colon , \pi^{k-1} \rangle$, or the space of unsolved environments at level k. Then

$$\begin{aligned} R(\langle \pi^1 \colon \pi^{k-1}, \pi \rangle) &= \int_X l(x, \langle \pi^1 \colon \pi^{k-1}, \pi \rangle) p(x) dx \\ &= \int_{X|\neg 1\colon \neg(k-1)} l(x, \langle \pi^1 \colon \pi^{k-1}, \pi \rangle) p(x) dx \end{aligned} \tag{14}$$

It is only the space of unsolved environments that appear in the risk at level k. The environments already correctly classified are of no concern.

3 Implementation

3.1 Train

The data $\mathcal{D}$ consists of N environments x_i. We run elements of the library $\mathcal{L}$ on each environment, resulting in L costs c_{ij} for x_i. This gives us information about the performance of all elements over a number of environments. A further step is to extract features $\phi \in \mathbb{R}^d$ from the data. As in other learning procedures, the features encode the information relevant to prediction. The scoring function $s(x, \xi)$ was referred to in an abstract form in Section 2. Here we make the scorer concrete. It is chosen to be a linear function of the features ϕ. We discuss how to solve loss-sensitive classification without referencing k, the list level. There are two cases for the features.

3.1.1 $\phi = \phi(x, \xi)$

Features are computed on an environment-element pair. There is unique information for each element in the context of an environment. $\phi_{ij} = \phi(x_i, \xi_j)$ is an abbreviation. $\hat{\phi}_{ij} = \phi_{ij} - \frac{1}{L}\sum_{l=1}^{L} \phi_{il}$ are the centered features, required for (6), the constraint that scores sum to zero. Scores are linear in features, $s_{ij} = \beta^T \hat{\phi}_{ij}$, $\beta \in \mathbb{R}^d$. We choose two standard convex forms for ψ: hinge, $\psi(t) = (1 + t)_+$, and square, $\psi(t) = (1 + t)^2$. Plugging all terms into the expression for the empirical surrogate risk (8a), the optimization problems are

$$J(\beta) = \sum_{i=1}^{N} \sum_{j=1}^{L} l_{ij}(1 + \beta^T \hat{\phi}_{ij})_+ + \frac{\lambda}{2}\beta^T\beta \tag{15a}$$

$$J(\beta) = \sum_{i=1}^{N} \sum_{j=1}^{L} l_{ij}(1 + \beta^T \hat{\phi}_{ij})^2 + \frac{\lambda}{2}\beta^T\beta \tag{15b}$$

$$\min_{\beta \in \mathbb{R}^d} J(\beta) \tag{15c}$$

$J(\beta)$ is the sum of the empirical surrogate risk and a regularization term. λ is the regularization weight and controls for overfitting the training data. In binary classification, using a hinge ψ leads to an SVM. The form in (15a) is similar to an SVM. In binary classification, a square ψ leads to regression. Similarly, the form in (15b) corresponds to loss-weighted regression. We try to regress from the features to the losses, but focus on cases where losses l_{ij} are high.

3.1.2 $\phi = \phi(x)$

Features are computed on the environment only. We abbreviate $\phi_i = \phi(x_i)$. Since the features have no information about the library elements, the scorers use a set of weights, $\beta_j \in \mathbb{R}^d$, $j = 1{:}\ L$, one for each library element. $\hat{\beta}_j =$

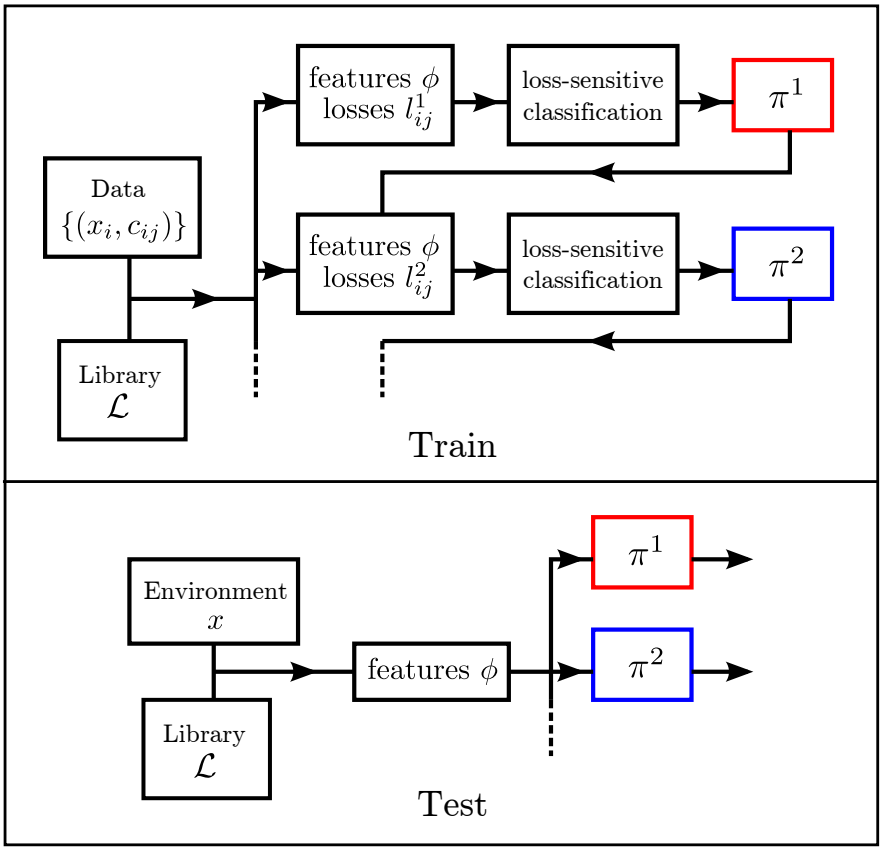

Figure 2: A flowchart of the train and test steps of list prediction.

$\beta_j - \frac{1}{L}\sum_{l=1}^{L}\beta_l$ are the centered weights, required for constraint (6). With this choice, $s_{ij} = \hat{\beta}_j^T \phi_i$. The optimization problems are

$$J(\beta_{1:L}) = \sum_{i=1}^{N}\sum_{j=1}^{L} l_{ij}(1 + \hat{\beta}_j^T \phi_i)_+ + \frac{\lambda}{2}\sum_{j=1}^{L}\beta_j^T\beta_j \tag{16a}$$

$$J(\beta_{1:L}) = \sum_{i=1}^{N}\sum_{j=1}^{L} l_{ij}(1 + \hat{\beta}_j^T \phi_i)^2 + \frac{\lambda}{2}\sum_{j=1}^{L}\beta_j^T\beta_j \tag{16b}$$

$$\min_{\beta_{1:L}\in\mathbb{R}^d} J(\beta_{1:L}) \tag{16c}$$

All the above optimization problems are convex optimization problems. In this paper, we solve them using the primal subgradient method. By introducing a superscript k into the losses, l_{ij}^k, and weights, β_j^k, in the above, we get the objectives, J^k, to be optimized at each level.

3.2 Test

After training, we obtain a list of scorers, which in turn defines a list of predictors, $\langle \pi^1 : \pi^B \rangle$. The scorers are used for prediction as in (13c). During testing, we are handed a query environment x. We compute features ϕ based on x and the library $\mathcal{L}$. The features are fed to each predictor, resulting in a list of elements $\langle \pi^1(x) : \pi^B(x) \rangle$. Training and testing for list prediction are summarized in Figure 2.

4 Discussion

4.1 Focus on unsolved environments

Rewriting the risk at level k as in (14), we see the mathematical reason for the intuition that deeper in the list, the focus is on unsolved environments. This is also clear from the optimization setup (15, 16). If x_i has been classified correctly before level k, $l_{ij}^k = 0 \, \forall \, j$, and x_i is ignored by the optimization.
The predictor space Π plays a role in performance. A weak space may not be able to deal with the unsolved environments. Increasing the length of the list will not help, as $X|\neg 1 \colon \neg(k-1)$ will not reduce significantly with k. On the other hand, a powerful space Π may achieve sufficiently low risk in the first level. The issue of Π is, however, orthogonal: predicting a list will never have higher risk than predicting a single element, and is always helpful.

4.2 Cost regression

Given the training data, an approach that comes to mind is to regress from features ϕ to costs c, and pick the element with the lowest cost. This procedure, cost regression, if successful, provides a lot of information. It would tell us the minimum cost that can be achieved by the library. It would also allow arbitrary addition of elements to the library.
In this work, we assume the library is given, and focus only on prediction. This allows us to deal with losses instead of costs. Cost regression is aiming for a more difficult task than necessary. The next approach that suggests itself is to regress from features ϕ to losses l. As seen in (15b), this is close, but not correct: the right thing to do is loss-weighted regression. With classification, we have not found scope for adding new elements to the library. When the library is modified, the predictor list must be trained again.

4.3 Element costs

Element costs may have to be preprocessed to capture the quantity of interest. For example, in trajectory seed optimization, a large finite cost may have to be assigned to 'failed' seeds. Similarly, if we consider any cost below a limit as 'solving' trajectory planning, thresholding is required. Once costs are decided, a fixed notion of loss is used throughout list prediction.

5 Case Study A: Seed Prediction for Trajectory Optimization of 2D Point Robot

5.1 Motivation

For the problem of planning a trajectory from a start to goal configuration, local trajectory optimization is an approach where an initial seed joining the start to goal is optimized. Since trajectory optimization is a non-convex problem, the solution quality is heavily dependent on the initial seed. Often these methods converge to a bad local minimum around the initial seed, e.g, passing through the middle of obstacles. The effectiveness of a seed is not known a priori. A computationally expensive approach is to optimize every element in a library of seed trajectories. List prediction can be used instead to predict a small set of elements.

5.2 Environment x and distribution $p(x)$

We consider a point robot in a 2D environment.
The environment is a unit square in $\mathbb{R}^2$ of dimensions $[0,1] \times [0,1]$. The start state is $(0,0)$ and the goal state $(1,1)$. The obstacles in the environment consist of 21 squares, each of side 0.1 units. 20 of these squares have centers sampled uniformly randomly from $[0.1, 0.9] \times [0.1, 0.9]$. 1 square is placed at $(0.5, 0.5)$ to ensure that there are no trivial cases where the straight line joining start and goal is collision free.

5.3 Element ξ

An element is a seed trajectory that connects the start to the goal. We represent a trajectory as a discretized set of waypoints. Thus element $\xi \in \mathbb{R}^{100\times 2}$, where the number of waypoints along the trajectory is 100 and the dimension of each state is 2.

5.4 Costs $c(x, \xi)$

The cost $c(x, \xi)$ is the cost of a seed trajectory ξ after it has been locally optimized by Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [19].
CHOMP is an unconstrained local trajectory optimization method. It optimizes the following objective function.

$$G(\xi) = G_{\text{smooth}} + \lambda_{\text{obs}} G_{\text{obs}} \tag{17}$$

where the first term, G_{smooth}, measures smoothness of the trajectory while the second term, G_{obs} measures proximity to obstacles. The term λ_{obs} is a scaling factor.

The smoothness cost G_{smooth} is expressed as

$$G_{\text{smooth}} := \text{tr}\left(0.5\,\xi^T\mathbf{A}\xi + \xi^T\mathbf{B} + \mathbf{C}\right) \tag{18}$$

The smoothness matrix terms, $\mathbf{A} \in \mathbb{R}^{100\times100}$, $\mathbf{B} \in \mathbb{R}^{100\times2}$, and $\mathbf{C} \in \mathbb{R}^{2\times2}$ are,

$$\begin{aligned}\mathbf{A} &:= \begin{bmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix}, \\ \mathbf{B} &:= \begin{bmatrix} -v & \mathbf{0} & \dots & \mathbf{0} & -w \end{bmatrix}^T,\end{aligned} \tag{19}$$

and

$$\mathbf{C} := 0.5\left(vv^T + ww^T\right), \tag{20}$$

where $v = (0,0)^T$ and $w = (1,1)^T$ are the start and goal states respectively. The obstacle cost is G_{obs} is expressed as

$$G_{\text{obs}}(\xi) := \sum_{i=1}^{99} w_{\text{obs}}(\xi_i)\,||\xi_{i+1} - \xi_i||_2 \tag{21}$$

where ξ_i is the i-th waypoint on the discretized path and the weight, $w_{\text{obs}}(x)$, is given by

$$w_{\text{obs}}(x) := \begin{cases} 0 & \text{if } \delta(x) > \epsilon \\ 0.5\left(\epsilon - \delta(x)\right)^2/\epsilon & \text{if } \epsilon \geq \delta(x) \geq 0 \\ 0.5\epsilon - \delta(x) & \text{if } \delta(x) < 0, \end{cases} \tag{22}$$

where ϵ is a tuning parameter defining obstacle clearance. The function $\delta(x)$ is the distance from the state to the nearest obstacle boundary and is negative when the state is inside an obstacle.
The cost function (18) is not convex due to G_{obs} and has multiple local minima. A local optimization is guaranteed to only converge to a local minima, which depends on the seed trajectory used for optimization.
CHOMP is a quasi-Newton approach to iteratively optimize the path. The update equation of CHOMP is

$$\xi^{i+1} := \xi^i - \eta^i\mathbf{A}^{-1}\nabla G(\xi^i), \tag{23}$$

where i is the iteration index, η^i is the step size, $\nabla G(\xi^i)$ is the gradient and ξ^{i+1} is the updated trajectory .
CHOMP is executed till one of the following termination conditions are fulfilled

1. Maximum iteration limit reached.
2. Relative cost improvement limit is reached.

Table 1: Parameters used in CHOMP

Parameters	**Values**
Obstacle Cost Weight λ_{obs}	100.0
Obstacle Clearance ϵ	0.05
Maximum Iteration Limit	100
Relative Cost Improvement Limit	10^{-5}

The parameters of CHOMP are listed in Table 1.
Let the CHOMP operation be represented by $\xi^+ = \mathcal{O}(\xi)$ where ξ^+ is the converged trajectory. Then the cost of an element $c(x, \xi)$ is

$$c(x, \xi) = G(\mathcal{O}(\xi)) \tag{24}$$

5.5 Library $\mathcal{L}$

We generated a library consisting of a diverse set of seed trajectories. Note that library generation is a separate process from list prediction.
We implemented an iterative library addition process. Assume that at the current iteration, the library already contains $l-1$ elements. An environment is randomly sampled from $p(x)$. If the current library does not contain a seed which, when optimized, leads to an acceptable solution, an expensive global optimization routine is invoked. The resulting trajectory is then appended to the library as element l. Iterations are repeated till the size of the library is L.

5.6 Features ϕ

Features $\phi(x, \xi)$ are computed on a pair of environment and seed. The optimization problem for this case is (15).
The context $\phi(x, \xi)$ is a 73 dimension vector consisting of the following terms

1. Cost of the unoptimized seed, and the linear term in its expected decrement (2 dimensions).

2. The sub-sampled signed distance gradient, and its hessian trace (21 dimensions).

3. The sub-sampled cost around the trajectory, and its components with the gradient (50 dimensions).

5.7 Application Parameters

Table 2 lists the parameters.

Table 2: Application Parameters

Parameter	Value
Library Length L	39
List Budget B	3
Feature Dimension d	73
Train Data N	700
Validation Data	200
Test Data	100

Table 3: Seed Trajectory Prediction for 2D Point Robot

List Size	Hinge Loss	Square Loss
Single Element	0.1073	0.1106
3 Elements	0.0715	0.0721

5.8 Results

Table 3 shows the results.

The distribution $p(x)$ places large probability mass on environments where obstacles are clustered. The predictor at the first level predicts simple seeds that go around these clusters. But there are environments which require the optimal seed to be in a complicated homotopy class. These are infrequent under $p(x)$, so they are ignored by the first predictor. Subsequent predictors focus on these environments, making customized predictions. Figures 3 and 4 make this point with specific examples.

5.9 Loss regression

While Dey et al. [6] advocates the use of regression to marginal losses at each level on examples which were misclassified, we found that this performs worse than loss-weighted regression. This supports our discussion in Section 4.2. Table 4 shows the results of loss regression for the 2D point robot. Table 5 shows the results for the 7D manipulator.

Table 4: Regression in Seed Trajectory Prediction for 2D Point Robot

List Size	Loss Sensitive Classification	Loss Regression
Single Element	0.1073	0.1244
3 Elements	0.0715	0.0785

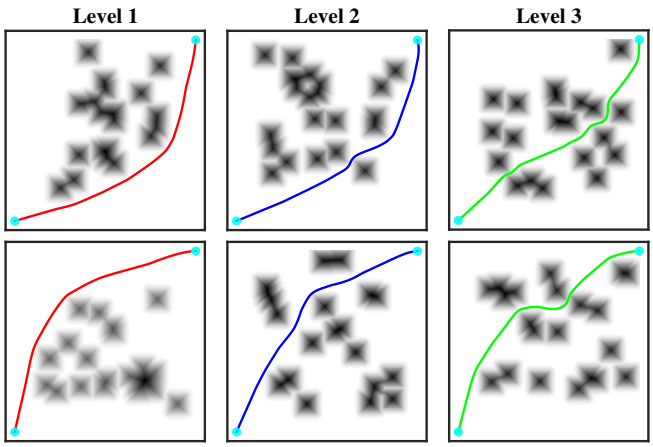

Figure 3: Training phase. The objective is to optimize a trajectory from start to goal (cyan dots) in an obstacle field (grayscale image). The examples shown are problems solved by predictors at different levels and the trajectories shown are post-optimization. The level 1 predictor (red) learns a simple classification rule to solve a large number of problems—it predicts seeds that simply go around a cluster of obstacles to achieve the lowest cost. The level 2 predictor (blue) focusses on and solves environments that level 1 did not solve. It learns to predict seeds in better homotopy classes. The level 3 predictor (green) focusses on corner cases, e.g the two instances shown have the optimal trajectory passing through a narrow gap that is surrounded by local minima. The level 3 predictor learns seeds that are optimized into this narrow gap.

Table 5: Regression in Seed Trajectory Prediction for 7D Manipulator

List Size	**Loss Sensitive Classification**	**Loss Regression**
Single Element	15.454	11.548
3 Elements	3.6085	6.3826

5.10 Propagating information down levels

In the presented framework, information is propagated down levels through marginal losses. We could also explicitly include information about which elements have been predicted so far. This can be done by using different features ϕ^k at each level. Note that this means that the space of predictors at each level, Π^k, is different. The successive predictor spaces intuitively have more information. However, this also means that the theoretical submodular guarantees on performance cannot be directly applied. An analysis for this case was presented in [17]. Consider level k, for an environment x. As in

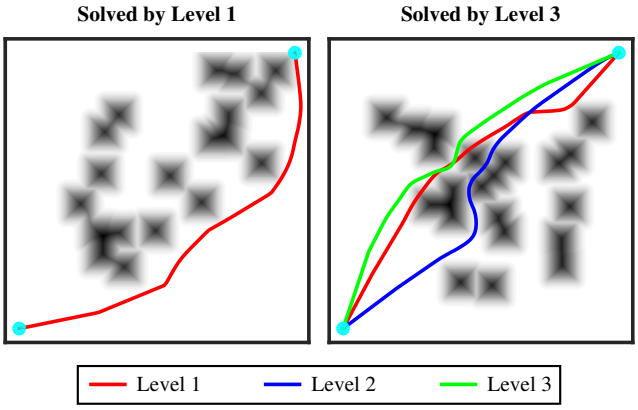

Figure 4: Test predictions. The environment on the left is solved by the level 1 predictor (red) which predicts an initial seed that goes around the obstacles on optimization. On the other hand, the environment on the right is solved only by level 3. The optimal trajectory passes through a narrow gap with a kink while there are many local minima surrounding this trajectory. Level 1 (red) makes a naive prediction that gets stuck cutting across obstacles. Level 2 (blue) comes closer to solving it but chooses a wrong homotopy class. Level 3 (green) solves the environment by predicting a seed which is optimized into the narrow gap.

[17], we define features at this level as

$$\begin{aligned}
\phi^k(x,\xi) &= (\phi(x,\xi)\ \phi_{\text{avg_diff}}\ \phi_{\text{min_diff}}) \\
\delta\phi^m(x,\xi) &= |\phi(x,\xi) - \phi(x,\xi^m)|, m = 1:(k-1) \\
\phi_{\text{avg_diff}} &= \frac{1}{k-1}\sum_{m=1}^{k-1}\delta\phi^m(x,\xi) \\
\phi_{\text{min_diff}} &= \min_m \delta\phi^m(x,\xi)
\end{aligned} \tag{25}$$

Appending features leads to better performance on our trajectory prediction datasets. Results are in Table 6. For both sets a budget of $B = 3$, and the hingle surrogate was used. The extra information leads to better performance, but this comes at the cost of a more complex implementation.

Table 6: Effect of appending features down levels

Dataset	**Same features down levels**	**Appended features down levels**
Point Robot	0.0715	0.0506
Manipulator	3.6085	3.5932

6 Case Study B: Seed Prediction for 7D Manipulator

6.1 Dataset

This dataset was taken from Dey et al. [6]. The task is conceptually similar to the seed prediction problem considered in Section 5. A seed from start to goal is optimized. In place of a 2D point robot is a 7D manipulator. The environment consists of obstacles on a tabletop. We ran our implementation of list prediction on this dataset for completeness. We omit further details on costs and features, which may be found in the reference.

6.2 Application Parameters

Table 7 lists the parameters.

Table 7: Application Parameters

Parameter	**Value**
Library Length L	30
List Budget B	3
Feature Dimension d	17
Train Data N	310
Validation Data	112
Test Data	100

6.3 Results

Table 8: Seed Trtajecotry Prediction for 7D Manipulator

List Size	**Hinge Loss**	**Square Loss**
Single Element	15.454	13.013
3 Elements	3.6085	3.6799

Table 8 shows the results.

7 Case Study C: Heuristic Prediction in Search Based Planning

7.1 Motivation

Heuristics are essential to improving the runtime performance of search based planning. The original A* paper [9] demonstrated that by using heuristic functions as an estimate of cost-to-go, the number of node expansions of the search algorithm were dramatically reduced in practice. However a heuristic function was required to be admissible for A* search to guarantee finding the optimal path.

Weighted A* [16] introduced the concept of inflated heuristics—functions that were allowed to overestimate the cost-to-go to the goal by a factor. Using such functions improved the runtime performance of the algorithm dramatically, while still allowing theoretical bounds on the sub optimality of the solution.

In the domain of motion planning, it is non-trivial to define good estimates of the cost-to-go. This is because the cost-to-go function is required to reason about the distribution of obstacles without being too computationally expensive to evaluate.

Recent approaches such as MHA* [1] have altered the way heuristics are viewed. Instead of being estimates of cost-to-go, heuristic functions can be viewed as *ranking functions* that rank the open queue in A* search. In this paradigm, the heuristic function simply penalizes each state in the open queue and the state with the least penalty is expanded. The notion of penalty in this paradigm is a measure of how unlikely is it that a state is part of the optimal solution.

Such heuristic functions may lose the nature of admissibility, or may not even be a weighted inflation of the true cost-to-go. The contribution of MHA* in this case is that it allows the use of such a heuristic function as long as it is anchored by a conventional weighted A*. MHA* allows heuristics to have a lot of flexibility and effectively act as modules that expand promising states only.

The motivation behind heuristic prediction is that there is no 'one size fits all' heuristic function. Depending on the environment, different heuristic functions will be effective. Given a library of inadmissible heuristics, list prediction can be used to predict a small subset of heuristics.

7.2 Environment x and distribution $p(x)$

We consider a 4 link arm in a 2D environment. The first joint of the arm is pivoted at the origin $(0, 0)$. Joint i can attain an angle $\theta_i \in (-\pi, \pi]$. The length of each link is 0.075 units.

The start state of the arm is $(0, 0, 0, 0)$. The goal state of the arm is $(\pi, 0, 0, 0)$.

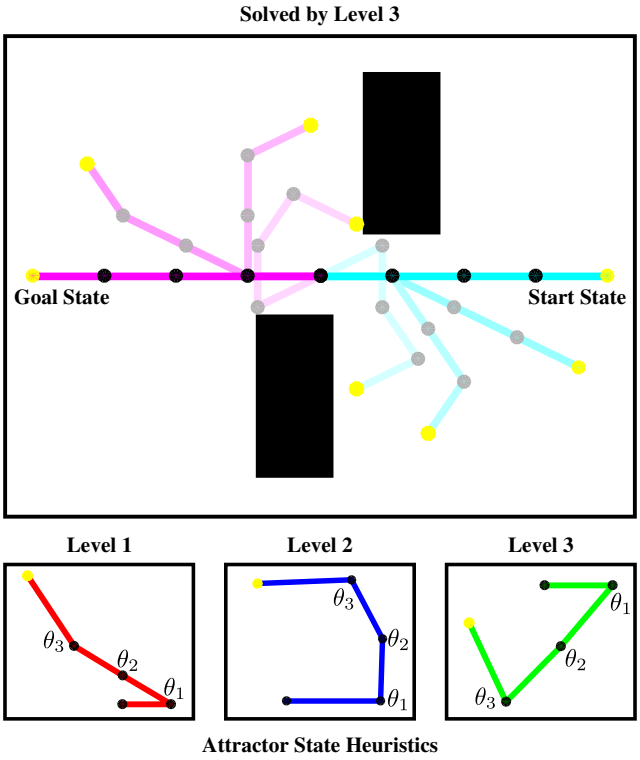

Figure 5: Sample heuristic list predicted for Section 7. MHA* is used to plan a trajectory for a 4 link arm from a start state (cyan) to goal state (magenta). The environment has two blocks creating a gap. To pass through, the arm has to tuck in and roll out again. Heuristics used are attractor states. The figure shows an infrequent environment for which the gap is narrow. The predictor at level 1 predicts a heuristic (red) that naively tucks the arm in the direction of the gap. However the upper block hinders this approach. The predictor at level 2 predicts a heuristic (blue) that brings the end effector closer to the base joint, however, the gap is small enough that this too is ineffective. The predictor at level 3 predicts a heuristic (green) that tucks the arm in a non-trivial configuration that allows it to pass through the gap. Snapshots from the trajectory resulting from the third heuristic are shown.

The environment x is a bounding box $[-0.4, 0.4] \times [-0.3, 0.3]$. It consists of 2 rectangular blocks, one above and one below the arm. The rectangular blocks have width 0.075 and height 0.2. The y coordinate of the upper and lower blocks is 0.15 and -0.15. The x coordinate of both blocks is sampled uniformly randomly from $[-0.225, 0.225]$.

Figure 5 shows an example of such an environment.

7.3 Element ξ

An element is an inadmissible heuristic function.
A heuristic function $h : \theta \to \mathbb{R}$ maps the joint configuration θ to a scalar value. In this experiment we use as heuristic an exponential kernel on a chosen state, called an 'attractor state'. This is expressed as

$$h(\theta, \theta^d) = \frac{1}{2} e^{(\theta_1 - \theta_1^d)^2 + (\theta_2 - \theta_2^d)^2 + (\theta_3 - \theta_3^d)^2} \tag{26}$$

where θ^d is an 'attractor state'. The heuristic penalizes arm states away from the attractor state.
The intuition behind using such heuristic functions is as follows. To go from start to goal, the 4 link arm has to be 'tucked' in special manners such that it can pass through the gap between the two rectangular blocks. The effectiveness of a tucking configuration depends on the location and width of the two blocks.
Each heuristic function from the specified family of heuristic functions prefers the arm to be similar to an 'attractor' state, θ^d. For example an attractor state $(0, 0, 0)$ prefers the arm to be outstretched.
Note that the heuristic is invariant to base joint angle θ_0. We found in our experiments that this invariance was important.

7.4 Costs $c(x, \xi)$

The element ξ is used as a heuristic input to MHA*. MHA* plans on a lattice created by discretizing each joint space 12 times uniformly from $(-\pi, \pi)$. Thus the number of vertices of the graph is 12^4. Each state has a connectivity of 2^4.
$c(x, \xi)$ is set to be the number of states expanded when using ξ. This value is saturated at a maximum of 10000 expansions. The cost is scaled from 0 to 10.

7.5 Library $\mathcal{L}$

A library of heuristics is generated by randomly sampling 99 attractor states. Each state of the attractor state is sampled uniformly randomly from $(-\pi, \pi)$ We also add an element to the library corresponding to no heuristic. In this setup, MHA* reverts to weighted A*.

7.6 Features ϕ

Features $\phi(x)$ are computed on the environment only. The optimization problem for this case is (16).
ϕ is a vector of Histogram of Gradients on the image of the environment. Each environment is converted to an image where 1 pixel maps to 1^{-3} units. Hence the dimension of the image is 800×600 pixels. The histogram of

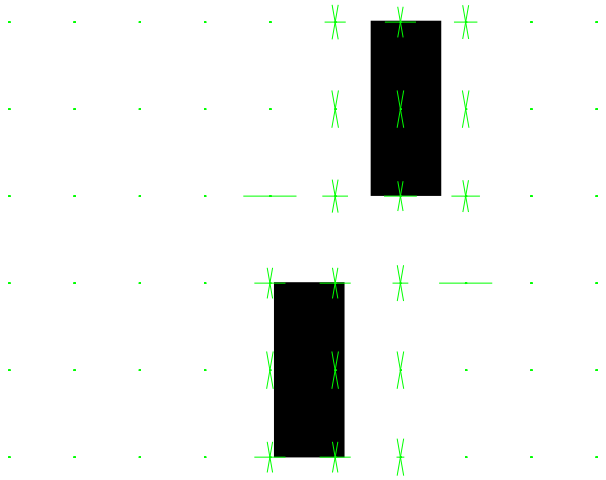

Figure 6: Histogram of Gradients (HoG) features extracted from the image.

gradient uses a cell size of 75×100 pixels. The length of the feature vector is 1620. An illustration is shown in Figure 6.

7.7 Application Parameters

Table 9 lists the parameters.

Table 9: Application Parameters

Parameter	**Value**
Library Length L	101
List Budget B	3
Feature Dimension d	1620
Train Data N	675
Validation Data	193
Test Data	96

7.8 Results

Table 10 shows the results.
Under $p(x)$, environments frequently have a sufficient gap between the two blocks for the arm to pass through. The predictor at the first level predicts attractor states corresponding to simple arm 'tucking' configurations. Environments in which the blocks are close together, leading to a narrow gap,

Table 10: Heuristic Prediction

List Size	**Hinge Loss**	**Square Loss**
Single Element	0.0976	0.0933
3 Elements	0.0325	0.0360

are infrequent. These environments require a complicated 'tucking' attractor state. The subsequent predictors solve such environments, as seen in Figure 5.

8 Case Study D: Planner Prediction in Adpative Motion Planning

8.1 Motivation

Real-time motion planning is a hard problem. It is difficult to design such a system that produces high quality solutions under time constraints in all situations.
The components of the motion planning system are—the planning algorithm, parameters of the planning algorithm, heuristic used by the algorithm, graph creation or sampling techniques used by the algorithm, etc.
The effectiveness of a combination of such parameters depends on the configuration of obstacles. As a result, predicting a list of potential planners given a planning problem leads to improved performance. The notion of a list of planners to create a planner ensemble has shown promising results [5, 4].

8.2 Environment x and distribution $p(x)$

We consider a point robot in a 2D environment.
The start state is $(0, 1000)$ and the goal state $(2000, 1000)$.
The environment is a square in $\mathbb{R}^2$ of dimensions $[0, 2000] \times [0, 20000]$. The obstacles in the environment are 2D circular discs. For a given environment, the radius of circular discs are fixed and sampled uniformly from $[10, 150]$. The number of discs are chosen from a spatial Poisson distribution with mean 10^{-5}. The centre of discs are distributed uniformly randomly. Additionally a circular disc of radius 7 is added at $(1000, 1000)$. See Figure 7 for a sample environment.

8.3 Element ξ and library $\mathcal{L}$

Each element is a sampling-based motion planning algorithm. A library of such algorithms is generated by varying tree growing strategies, sampling strategies and heuristics.
Table 11 shows a table of such planners. The planners are planning algorithms from OMPL (Open Motion Planning Laboratory) with a specific configuration of parameters.

Table 11: Planner database of 100 planners

Algorithm	**Parameter**	**Config 1**	**Config 2**	**Config 3**	**Config 4**	**Config 5**	**Config 6**	**Config 7**	**Config 8**	**Config 9**	**Config 10**
RRT	Range	2.0×10^2	4.0×10^2	6.0×10^2	8.0×10^2	1.0×10^3	1.2×10^3	1.4×10^3	1.6×10^3	1.8×10^3	2.0×10^3
RRT-Connect	Range	2.0×10^2	4.0×10^2	6.0×10^2	8.0×10^2	1.0×10^3	1.2×10^3	1.4×10^3	1.6×10^3	1.8×10^3	2.0×10^3
EST	Range	2.0×10^2	4.0×10^2	6.0×10^2	8.0×10^2	1.0×10^3	1.2×10^3	1.4×10^3	1.6×10^3	1.8×10^3	2.0×10^3
T-RRT	Range	2.0×10^2	2.0×10^3	2.0×10^2	2.0×10^2	2.0×10^2	2.0×10^2	2.0×10^3	2.0×10^3	2.0×10^3	2.0×10^3
	Temperature (T)	2.0×10^0	2.0×10^0	1.0×10^{-1}	1.0×10^{-1}	1.0×10^1	1.0×10^1	1.0×10^{-1}	1.0×10^{-1}	1.0×10^1	1.0×10^1
	Frontier Ratio (ρ)	1.0×10^{-1}	1.0×10^{-1}	1.0×10^{-2}	1.0×10^0	1.0×10^{-2}	1.0×10^0	1.0×10^{-2}	1.0×10^0	1.0×10^{-2}	1.0×10^{-2}
LBT-RRT	Range	2.0×10^3	2.0×10^2	2.0×10^2	2.0×10^2	1.0×10^3	1.0×10^3	1.0×10^3	2.0×10^3	2.0×10^3	2.0×10^3
	Approximation (ε)	4.0×10^{-1}	5.0×10^{-3}	5.0×10^{-2}	5.0×10^{-1}	5.0×10^{-3}	5.0×10^{-2}	5.0×10^{-1}	5.0×10^{-3}	5.0×10^{-2}	5.0×10^{-1}
RRT*	Range	2.0×10^3	2.0×10^2	2.0×10^2	2.0×10^2	1.0×10^3	1.0×10^3	1.0×10^3	2.0×10^3	2.0×10^3	2.0×10^3
	Radius Factor (γ)	1.0×10^0	1.0×10^0	3.0×10^0	1.0×10^1	1.0×10^0	3.0×10^0	1.0×10^1	1.0×10^0	3.0×10^0	1.0×10^1
Informed RRT*	Radius Factor (γ)	1.0×10^0	2.0×10^0	3.0×10^0	4.0×10^0	5.0×10^0	6.0×10^0	7.0×10^0	8.0×10^0	9.0×10^0	1.0×10^1
RRT* Tunnel	Range	2.0×10^3	2.0×10^2	2.0×10^2	2.0×10^2	1.0×10^3	1.0×10^3	1.0×10^3	2.0×10^3	2.0×10^3	2.0×10^3
	Radius Factor (γ)	1.0×10^0	1.0×10^0	3.0×10^0	1.0×10^1	1.0×10^0	3.0×10^0	1.0×10^1	1.0×10^0	3.0×10^0	1.0×10^1
	Tunnel	2.0×10^2	2.0×10^2	5.0×10^1	5.0×10^2	5.0×10^1	5.0×10^2	5.0×10^1	5.0×10^2	5.0×10^1	5.0×10^2
RRT* Obstacle	Range	2.0×10^3	2.0×10^2	2.0×10^2	2.0×10^2	1.0×10^3	1.0×10^3	1.0×10^3	2.0×10^3	2.0×10^3	2.0×10^3
	Radius Factor (γ)	1.0×10^0	1.0×10^0	3.0×10^0	1.0×10^1	1.0×10^0	3.0×10^0	1.0×10^1	1.0×10^0	3.0×10^0	1.0×10^1
	Threshold	5.5×10^1	5.5×10^1	3.0×10^1	9.0×10^1	3.0×10^1	9.0×10^1	3.0×10^1	9.0×10^1	3.0×10^1	9.0×10^1
BIT*	Samples / Batch	4.0×10^3	1.0×10^2	1.0×10^2	1.0×10^2	1.0×10^3	1.0×10^3	1.0×10^3	8.0×10^3	8.0×10^3	8.0×10^3
	Radius Factor (γ)	1.0×10^0	1.0×10^0	3.0×10^0	1.0×10^1	1.0×10^0	3.0×10^0	1.0×10^1	1.0×10^0	3.0×10^0	1.0×10^1

8.4 Costs $c(x, \xi)$

The planner ξ is used to plan a trajectory within a time constraint of 0.05s. $c(x, \xi)$ is set to be equal to the path length of the solution. The cost is affinely transformed to $[0, 20]$. If no feasible path was found, $c(x, \xi)$ is set to 20.

8.5 Features ϕ

Features $\phi(x)$ are computed on the environment only. The optimization problem for this case is (16).
ϕ is a vector of Histogram of Gradients on the image of the environment. Each environment is converted to an image where 1 pixel maps to 5 units. Hence the dimension of the image is 440×440 pixels. The histogram of gradient uses a cell size of 50×50 pixels. The length of the feature vector is 1764.

8.6 Application Parameters

Table 12 lists the parameters.

Table 12: Application Parameters

Parameter	**Value**
Library Length L	100
List Budget B	3
Feature Dimension d	1764
Train Data N	579
Validation Data	166
Test Data	82

8.7 Results

Table 13: Planner Prediction

List Size	**Hinge Loss**	**Square Loss**
Single Element	0.2222	0.2281
3 Elements	0.0222	0.0281

Table 13 shows the results.
The first predictor predicts planners such as BIT*, RRT-Connect and Informed-RRT*. These planners don't make strong assumptions about structure in the environment, which results in good performance over a wide range of environments. Environments with structure are infrequent under $p(x)$. We

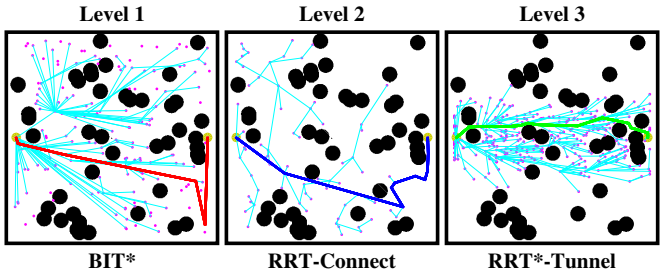

Figure 7: Sample lists predicted for Section 8. A planning algorithm is used to plan a trajectory from start to goal (beige). The environment consists of random circular obstacles. In this environment, the goal happens to be blocked off by a wall of 3 obstacles. The predictor at level 1 naively predicts BIT* [8], given its effectiveness in solving frequently occuring environments. However, the wall makes it difficult for BIT* to find a good solution in the time budget. The predictor at level 2 predicts RRT-Connect [11], given its effectiveness on environments where BIT* fails—however this too cannot plan around the wall. The predictor at level 3 predicts RRT*-Tunnel [5], as it concentrates sampling in a tunnel around the initial straight-line solution, and finds a path through the gap in the wall.

observe that subsequent predictors predict planners which exploit structure. See Figure 7.

9 Related Work

With a formulation for list prediction in place, we can discuss related work in a common language. Jetchev and Toussaint [10] was an early work on predicting seeds for trajectory planning. Cost regression and classification were implemented, without the formalism of loss-sensitive classification. Their results support discussion 4.2 that cost regression is a more difficult task. Dragan et al. [7] predicted the usefulness of end-effector goals for trajectory planning on a manipulator. Their work did not use a library of elements. However, some heuristics in their procedures are explained when a surrogate convex loss is used to derive algorithms. Both [10] and [7] predicted a single element. While a stronger predictor class Π was suggested to increase performance, a list was not.

ConseqOpt for list prediction appeared in Dey et al. [6]. The algorithm was presented along with a theoretical guarantee. Our work investigates the prediction step deeper. While regression was used for loss-sensitive classification in [6], our discussion 4.2 points out that loss-weighted regression is what follows from the surrogate[2]. Compared to [6], we also investigate the behavior of list prediction in motion planning more thoroughly. A follow up to ConseqOpt by Ross et al. [17] considered list prediction in a different setting. It was shown that when trained online, with data streaming in, a single predictor for predicting a list works well.

We believe the concepts of libraries, loss-sensitive classification and list prediction will be of benefit to existing work in motion planning. Specifically, they will benefit applications that allow multiple predictions to be made and evaluated, either sequentially or in parallel.

One area is trajectory prediction. Zucker [20] generates a 'behavior library' of optimized trajectories and predicts the best trajectory given a query. Berenson et al. [3] generates a library of past plans and uses a heuristic to select one that can be repaired easily to solve a new environment. Pan et al. [15] predicts if a seed trajectory will be successful for local optimization. Poffald et al. [12] uses a library of motion primitives and predicts the best primitive that can be adapated for a new environment. List prediction fits seamlessly into all these frameworks.

Wzorek et al. [18] predicts a motion planning strategy, from a library, that can be applied to repair a plan. Palmieri and Arras [14] learns a distance metric for an RRT to predict the nearest neighbour. Morales et al. [13] predicts a motion planner to apply to different sections of a planning problem. Even in these applications, it is possible to use list prediction for improving performance.

[2] We run experiments on the dataset from [6] for seed prediction on a 7D manipulator. Results are in Table 8.

10 Conclusion

We have presented comprehensive arguments advocating list prediction. We set up an unambiguous framework, developed intuition, and demonstrated results on a variety of planning problems. To establish a firm base, each step in list prediction that we considered is justified. There are modifications which can improve performance in practice. One of these strategies is to append features down a list, which we evaluated on one of our datasets, in Section 5.10.

List prediction can only do as well as the best element in the library. Building the set $\mathcal{L}$, or library generation, is higher than list prediction in the hierarchy of decision making. The simplest method of library generation is to sample the space of elements, as in our heuristics library, Section 7.5. It may be beneficial generate the library more intelligently. Our trajectory seeds library, Section 5.5, consisted of optimal trajectories in environments drawn from $p(x)$. A formal investigation into library generation is an interesting direction for future work.

11 Acknowledgement

Sanjiban Choudhury acknowledges the support from ONR grant N000141310821. Abhijeet Tallavajhula was supported by the National Science Foundation under Grant Number 1317803. The authors particularly thank Debadeepta Dey for a number of discussions, and sharing the grasp seed prediction dataset.

References

[1] Sandip Aine, Siddharth Swaminathan, Venkatraman Narayanan, Victor Hwang, and Maxim Likhachev. Multi-heuristic a*. In *Seventh Annual Symposium on Combinatorial Search*, 2014.

[2] Bernardo Ávila Pires, Csaba Szepesvari, and Mohammad Ghavamzadeh. Cost-sensitive multiclass classification risk bounds. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1391–1399, 2013.

[3] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. A robot path planning framework that learns from experience. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3671–3678. IEEE, 2012.

[4] Sanjiban Choudhury, Sankalp Arora, and Sebastian Scherer. The planner ensemble and trajectory executive: A high performance motion planning system with guaranteed safety. In *AHS 70th Annual Forum, Montreal, Quebec, Canada*, May 2014.

[5] Sanjiban Choudhury, Sankalp Arora, and Sebastian Scherer. The planner ensemble: Motion planning by executing diverse algorithms. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 2389–2395, 2015.

[6] Debadeepta Dey, Tian Yu Liu, Martial Hebert, and J Andrew Bagnell. Contextual sequence prediction with application to control library optimization. *Robotics*, page 49, 2013.

[7] Anca Dragan, Geoffrey J Gordon, and Siddhartha Srinivasa. Learning from experience in manipulation planning: Setting the right goals. 2011.

[8] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. *arXiv preprint arXiv:1405.5848*, 2014.

[9] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, 1968.

[10] Nikolay Jetchev and Marc Toussaint. Fast motion planning from experience: trajectory prediction for speeding up movement generation. *Autonomous Robots*, 34(1-2):111–127, 2013.

[11] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[12] Yajia Zhang Marcela Poffald and Kris Hauser. Learning problem space metrics for motion primitive selection. In *IROS, 2014*. IEEE, 2014.

[13] Marco Morales, Lydia Tapia, Roger Pearce, Samuel Rodriguez, and Nancy M Amato. A machine learning approach for feature-sensitive motion planning. In *Algorithmic Foundations of Robotics VI*, pages 361–376. Springer, 2005.

[14] Luigi Palmieri and Kai O Arras. Distance metric learning for rrt-based motion planning with constant-time inference. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 637–643. IEEE, 2015.

[15] Jia Pan, Zhuo Chen, and Pieter Abbeel. Predicting initialization effectiveness for trajectory optimization. In *Robotics and Automation, 2014. ICRA'14. IEEE International Conference on*. IEEE.

[16] Judea Pearl. Heuristics: intelligent search strategies for computer problem solving. 1984.

[17] Stephane Ross, Jiaji Zhou, Yisong Yue, Debadeepta Dey, and J Andrew Bagnell. Learning policies for contextual submodular prediction. *arXiv preprint arXiv:1305.2532*, 2013.

[18] Mariusz Wzorek, Jonas Kvarnström, and Patrick Doherty. Choosing replanning strategies for unmanned aircraft systems. 2010.

[19] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant hamiltonian optimization for motion planning. *The International Journal of Robotics Research*, 32(9-10):1164–1193, 2013.

[20] Matthew Zucker. A data-driven approach to high level planning. Technical Report CMU-RI-TR-09-42, Robotics Institute, Pittsburgh, PA, January 2009.